VISIÓN ARTIFICIAL

Entrega final

Alumno: José Antonio Nicolás Navarro
Curso: 4º
Correo: joseantonio.nicolasn@um.es
Profesor: Alberto Ruiz García y Pedro E. López de Teruel Alcolea

Índice

Mejoras con respecto a entrega parcial

Al parecer, no haber entregado la entrega parcial junto a una memoria (que no se pedía en la tarea, por cierto) no merecía ejecutar mi código para comprobar su funcionamiento, por lo que el feedback que recibí fue que me pusiera con ello, sin más indicaciones sobre las virtudes o deficiencias de mis programas.

Así que las mejoras son la propia documentación, y una corrección del ejercicio del cálculo del FOV, donde había un error por no pasar la medida de ángulos a radianes antes de introducirlos en la tangente. He realizado además 3 ejercicios opcionales extra; 'ANON', 'DLIB' y 'NFACE'.

Nota: no he incluido el archivo 'shape_predictor_68_face_landmarks.dat' por ser demasiado pesado, pero es necesario que se encuentre en la carpeta 'dlib' que está dentro de 'scripts y ejemplos' para que funcione el ejercicio 'NFACE' y 'DLIB'.

1. Ejercicio “CALIBRACIÓN”

a) Calibración de la cámara mediante múltiples imágenes de un *chessboard*

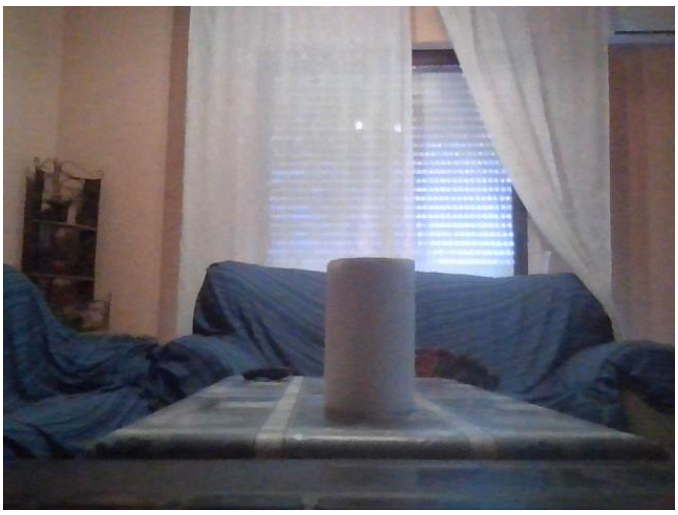
Para calibrar la cámara de mi equipo hice uso del script ‘calibrate.py’. De la matriz que se obtiene en la salida interesa sobre todo el primer valor, que indica la distancia focal *f*.

```
(base) C:\Users\janic\Desktop>python calibrate.py .\calibrate\*
processing .\calibrate\20210409-104727.png...
ok
processing .\calibrate\20210409-104738.png...
ok
processing .\calibrate\20210409-104800.png...
ok
processing .\calibrate\20210409-104810.png...
ok
processing .\calibrate\20210409-104821.png...
ok
processing .\calibrate\20210409-104831.png...
ok
RMS: 0.5098338826264498
camera matrix:
 [[680.71882429   0.         316.7311621 ]
 [  0.         571.83241935 222.65079013]
 [  0.           0.           1.        ]]
distortion coefficients:  [-6.47290439e-01  1.23154002e+01  1.16717432e-01  5.66004146e-03
 -6.38371708e+01]
```

Esta distancia como puede observarse es igual a 680.7188.

b) Calibración aproximada con un objeto y comparación con el resultado anterior.

El objeto escogido para la calibración aproximada y la toma de medidas es un rollo de papel como el que aparece en la siguiente imagen, también utilizada para la medida de su altura en píxeles:



- Cálculo de *f*:

X = altura del objeto = 22 cm
Z = distancia al objeto desde la cámara = 97 cm
u = altura en píxeles del objeto = 149 píxeles (obtenida desde "Recortes" de Windows)

$$u = f\frac{X}{Z} \Rightarrow f = \frac{u * Z}{X} = \frac{149 * 97}{22} = 656.95$$

Como puede observarse, la *f* calculada de forma aproximada es bastante similar a la obtenida mediante el script de calibración.

- Cálculo del FOV horizontal y vertical:

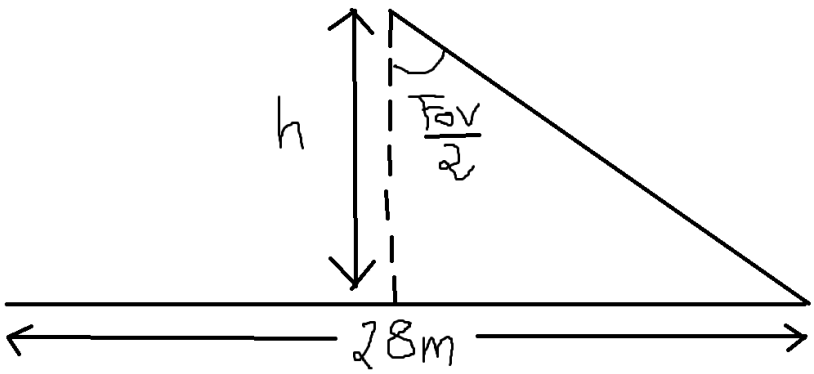
El cálculo del campo de visión, tanto vertical como horizontal viene dado por las siguientes expresiones matemáticas:

$$tan(\frac{FOV_{hor}}{2}) = \frac{w/2}{f} \Rightarrow FOV_{hor} = 2 * \arctan(\frac{w/2}{f}) = 2 * arctan(\frac{640/2}{656.95}) = 51.94º$$

$$tan(\frac{FOV_{ver}}{2}) = \frac{h/2}{f} \Rightarrow FOV_{ver} = 2 * \arctan(\frac{h/2}{f}) = 2 * arctan(\frac{480/2}{656.95}) = 40.14º$$

c) Altura de la cámara para obtener una vista cenital completa de un campo de baloncesto.

Un campo de baloncesto mide 28 metros de largo por 15 metros de ancho. Se puede representar el problema, en el caso del FOV horizontal, de la siguiente forma (con el FOV vertical sería la misma idea):



Para calcular la altura, por tanto, se utiliza la siguiente fórmula:

$$h_1 = \frac{a/2}{tan\ \frac{FOV_{hor}}{2}} = \frac{28/2}{tan\ \frac{51.94º}{2}} = 28.74\ m$$

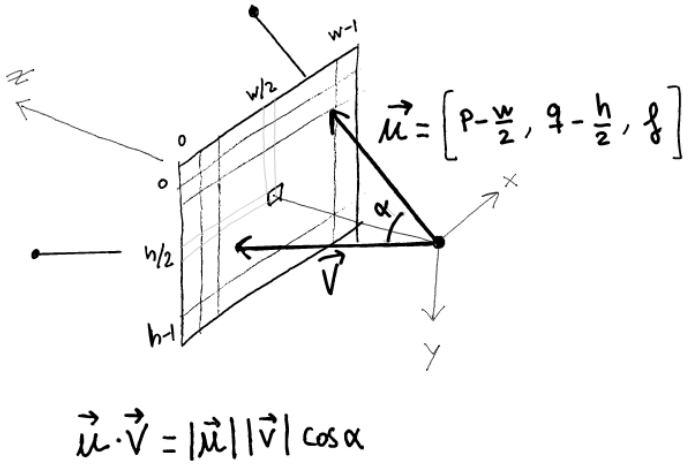
$$h_2 = \frac{b/2}{tan\ \frac{FOV_{ver}}{2}} = \frac{15/2}{tan\ \frac{40.14º}{2}} = 21.41\ m$$

Se obtienen dos alturas, pues existen dos campos de visión diferentes. Hay que quedarse con la altura más grande para que quepa todo el campo en la vista cenital, en este caso, h_1, que es igual a 28.74 m.

d) Aplicación para medir el ángulo que definen dos puntos marcados con el ratón en el imagen

El código de la aplicación dibuja círculos cada vez que se hace click, y cuando hay 2 puntos marcados se unen con una línea. Para facilitar las pruebas, si ya hay 2 puntos y se vuelve a hacer click, se reinicia para poder volver a tomar medidas nuevamente.

Aparte del dibujo y unión de los puntos, lo interesante del código es el cálculo de los grados a través de la distancia focal calculada previamente y el tamaño de la ventana que captura la cámara, despejando el ángulo de la siguiente expresión de los apuntes:

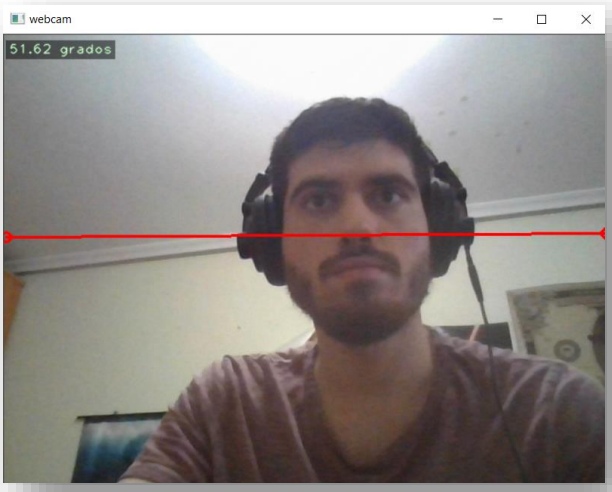


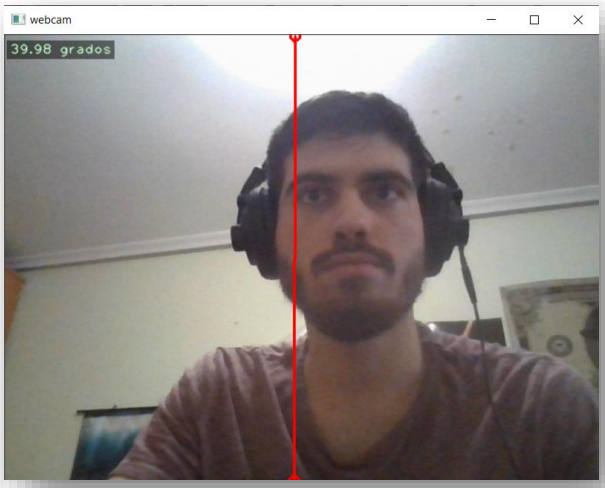
La parte del código que tiene interés y que mejor refleja lo que he explicado puede comprobarse a continuación:

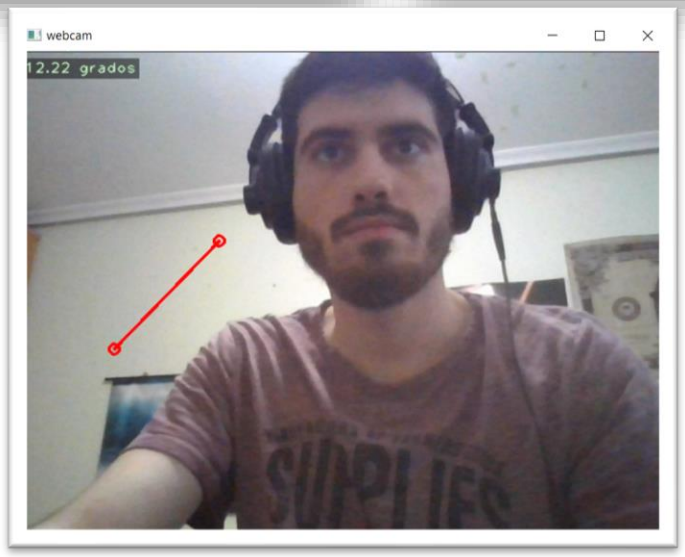
```
def fun(event, x, y, flags, param):
    global punto1, punto2
    if event == cv.EVENT_LBUTTONDOWN:
        if not punto1:
            punto1 = (x,y)
        elif not punto2:
            punto2 = (x,y)
        elif punto1 and punto2:
            punto2 = ()
            punto1 = (x,y)

cv.namedWindow("webcam")
cv.setMouseCallback("webcam", fun)

punto1 = ()
punto2 = ()

f = 656.95
w = 640
h = 480


def anguloEntrePuntos():
    u = (punto1[0] - w/2, punto1[1] - h/2, f)
    v = (punto2[0] - w/2, punto2[1] - h/2, f)

    prod_escalar = sum([i*j for i,j in zip(u,v)])
    modulos = (sum([i ** 2 for i in u]) ** 0.5) * (sum([i ** 2 for i in v
]) ** 0.5)

    return math.degrees(math.acos(prod_escalar / modulos))
```

Aclarar que, en cualquier caso, en este ejercicio y en todos los posteriores el código puede comprobarse más a fondo para su inspección puesto que ha sido adjuntado.

Para mostrar el funcionamiento de la aplicación, muestro tres ejemplos. En los dos primeros se toma todo el ancho y todo el alto para mostrar que el ángulo calculado se aproxima muchísimo a los campos de visión calculados anteriormente. El tercer ejemplo es una medida tomada aleatoriamente, mostrando el funcionamiento normal del programa.







2. Ejercicio “ACTIVIDAD”

2.1. Enunciado

Construye un detector de movimiento en una región de interés de la imagen marcada manualmente. Guarda 2 o 3 segundos de la secuencia detectada en un archivo de vídeo. Opcional: muestra el objeto seleccionado anulando el fondo.

2.2. Resolución

Para lograr detectar movimiento he utilizado la idea de guardarme un primer *frame* para tener siempre un *frame* actual y uno anterior de forma que, al realizar la diferencia absoluta entre los mismos, si se supera cierto umbral y los contornos de la imagen han cambiado lo suficiente para considerarse que ha habido movimiento entre ellos, se empieza a grabar. Además, se dibuja un círculo rojo indicando que la secuencia donde se detecta movimiento está siendo grabada.

El umbral donde se empieza a detectar movimiento (el valor de las áreas de los contornos) fue establecido en base a prueba y error hasta que solo se detectaban los movimientos evidentes y no movimientos muy leves.

Cuando se para el script, el vídeo resultante, que se encuentra en el mismo directorio donde está el archivo con el programa, contiene la secuencia de *frames* donde se ha detectado movimiento. El resto de detalles se muestra con el código de a continuación:

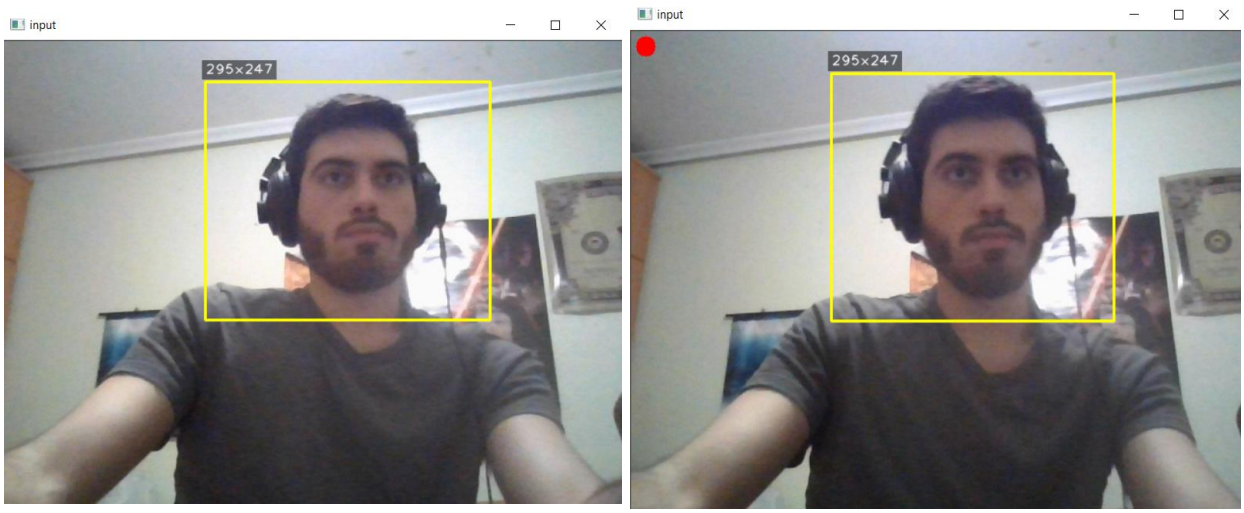
```
#!/usr/bin/env python

import numpy as np
import cv2 as cv
from datetime import datetime

from umucv.util import ROI, putText
from umucv.util import Video
from umucv.stream import autoStream

# Se crea la ventana que mostrará los frames capturados
cv.namedWindow("input")
cv.moveWindow('input', 0, 0)

region = ROI("input")
video = Video(fps=15)

frame_anterior = None
frames_counter = 0
```

```
# Se itera sobre cada frame capturado
for key, frame in autoStream():

    # Se convierte la imagen a escala de grises
    frame_actual = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    # Se suaviza la imagen
    frame_actual = cv.GaussianBlur(frame_actual, (21, 21), 0)

    # Se obtiene el primer frame y se salta al siguiente
    if frame_anterior is None:
        frame_anterior = frame_actual
        continue

    video.ON = False
    if region.roi:
        # Se obtienen las coordenadas de la región
        [x1,y1,x2,y2] = region.roi

        # Se realiza la diferencia absoluta
        resta = cv.absdiff(frame_anterior[y1:y2+1, x1:x2+1], frame_actual
[y1:y2+1, x1:x2+1])

        # Se guarda la diferencia a partir de un valor
        umbral = cv.threshold(resta, 25, 255, cv.THRESH_BINARY)[1]

        # Se dilata el umbral para tapar agujeros
        umbral = cv.dilate(umbral, None, iterations=2)

        # Buscamos contorno en la imagen
        contornos, hierarchy = cv.findContours(umbral.copy(), cv.RETR_TRE
E, cv.CHAIN_APPROX_SIMPLE)

        # Recorremos todos los contornos encontrados
        for c in contornos:

            # Eliminamos los contornos más pequeños
            if cv.contourArea(c) <= 1500:
                cv.circle(frame, (15,15), 10, (0, 0, 255), -1)
                # Se inicia la grabación de la ROI si se detecta movimien
to
                video.ON = True
                video.write(frame)
                break

        # Se dibuja un rectángulo en las coordenadas de la ROI mostrando
su tamaño
        cv.rectangle(frame, (x1,y1), (x2,y2), color=(0,255,255), thicknes
s=2)
```

```
        putText(frame, f'{x2-x1+1}x{y2-y1+1}', orig=(x1,y1-8))

    frame_anterior = frame_actual

    # Se dibuja cada frame captado por la webcam
    h,w,_ = frame.shape
    cv.imshow('input',frame)
```

2.3. Ejemplos

Seguidamente del código muestro unos ejemplos de funcionamiento. He adjuntado unas capturas donde, dentro de una región de interés, no se detecta movimiento en la imagen de la izquierda, y sí se detecta en la derecha:



Como este ejercicio se basa en la creación de vídeos, quizás el ejemplo no deja del todo claro el funcionamiento del programa y he creado una demostración que puede ser visualizada dentro de las carpetas adjuntadas.

3. Ejercicio “COLOR”

3.1. Enunciado

Construye un clasificador de objetos en base a la similitud de los histogramas de color del ROI (de los 3 canales por separado).

3.2. Resolución

La parte que pertenece realmente a la visión artificial en este ejercicio es la comparación de histogramas de una región de interés que esté siendo capturada con la lista de modelos añadidos a una lista para ser comparados.

Para ello, he hecho uso de una función que, dadas dos regiones de interés, calcula el histograma de ambas teniendo en cuenta los 3 canales del espacio de color y realizando una media, comparándolos utilizando la medida de error Chi cuadrado.

Para la región de interés actual, se itera sobre todos los modelos almacenados invocando a la función descrita y guardando aquel modelo con la menor diferencia con respecto a la ROI actual, que es el que acaba apareciendo finalmente en otra ventana, llamada ‘Detected’, mostrando el objeto o la región, por tanto, que más se asemeja. Adjunto prácticamente todo el código ya que nada me parece prescindible desde el punto de vista de la resolución del ejercicio:

```
# Función que compara los histogramas del frame actual y el del ROI y dev
uelve su comparación utilizando Chi-cuadrado
def comparaHistogramas(roiActual, roi):

    # Se calculan los histogramas para ambas ROIs teniendo en cuenta los
3 canales
    histogramaRoiActual = cv.calcHist([roiActual], [0, 1, 2], None, [8, 8
, 8], [0,256] + [0,256] + [0,256])
    histogramaRoi = cv.calcHist([roi], [0, 1, 2], None, [8, 8, 8], [0,256
] + [0,256] + [0,256])

    # Se normaliza el histograma
    histogramaRoiActual = histogramaRoiActual / np.sum(histogramaRoiActua
l)
    histogramaRoi = histogramaRoi / np.sum(histogramaRoi)

    return cv.compareHist(histogramaRoi, histogramaRoiActual, cv.HISTCMP_
CHISQR)


# Se crea la ventana que mostrará los frames capturados
```

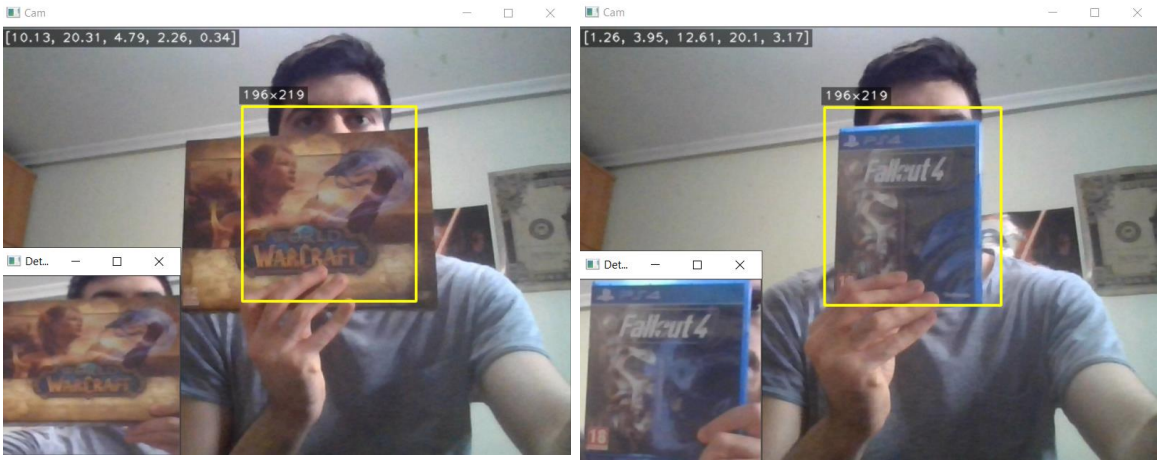
```
cv.namedWindow("Cam")
cv.moveWindow('Cam', 0, 0)
region = ROI("Cam")

limite = 5
modelos = []

# Se itera sobre cada frame capturado
for key, frame in autoStream():

    help.show_if(key, ord('h'))

    if region.roi:

        # Si hay un ROI se obtiene la imagen
        [x1,y1,x2,y2] = region.roi
        roiActual = frame[y1:y2+1, x1:x2+1]

        # Si se pulsa 'c' se añade la ROI a la lista de modelos
        if key == ord('c'):
            modelos.append(cv.resize(roiActual, (200,200)))
            modelosApilados = np.hstack(modelos)
            cv.imshow("Models", modelosApilados)

        # Si se pulsa 'x' se borran los modelos almacenados
        elif key == ord('x'):
            modelos.clear()
            cv.destroyWindow("Models")
            cv.destroyWindow("Detected")

        diferencias = []
        diferenciaMinima = (None, float('inf')) # la diferencia mínima in
icial es infinito

        # Se recorren los modelos para guardar el que más se parezca la R
OI actual y mostrarlo
        for m in modelos:
            diferencia = comparaHistogramas(roiActual, m)
            diferencias.append(round(diferencia,2))
            if diferenciaMinima[1] > diferencia:
                diferenciaMinima = (m, diferencia)

        putText(frame, str(diferencias))

        if diferenciaMinima[1] < limite:
            cv.imshow("Detected", diferenciaMinima[0])
        else:
            cv.destroyWindow("Detected")
```

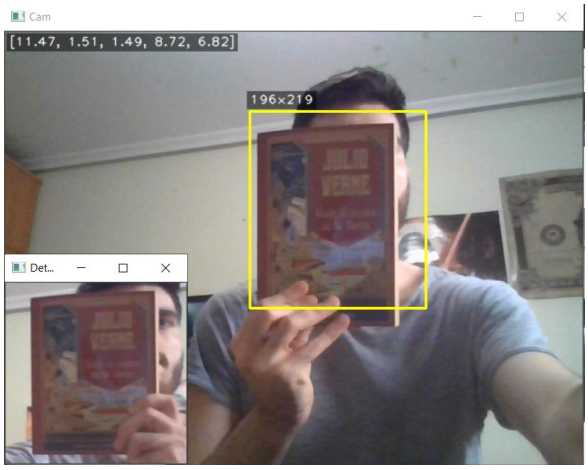
```
        # Se dibuja el rectángulo de la ROI
        cv.rectangle(frame, (x1,y1), (x2,y2), color=(0,255,255), thicknes
s=2)
        putText(frame, f'{x2-x1+1}x{y2-y1+1}', orig=(x1,y1-8))

    # Se dibuja cada frame captado por la webcam
    cv.imshow('Cam',frame)

cv.destroyAllWindows()
```

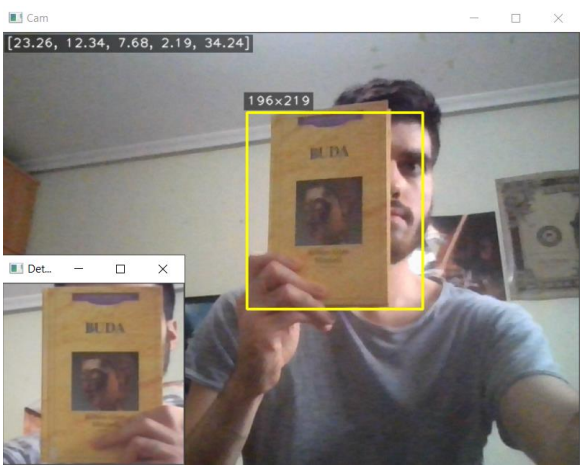
3.3. Ejemplos

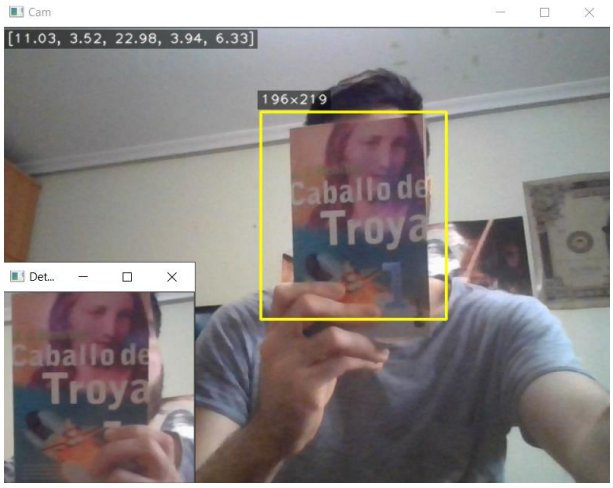
Para ilustrar el funcionamiento del programa, he tomado 5 objetos para clasificar; 3 libros y 2 carátulas de videojuegos. Cuando se selecciona una ROI, si se pulsa ‘c’, se añade la ROI a la lista de modelos. Conforme van guardándose, se añaden a la lista de modelos en una ventana separada, con el siguiente resultado final:

Con la lista de modelos construida para comparar los histogramas de color, se muestra en otra pequeña ventana el modelo con el que la ROI actual menos diferencia guarda, logrando los resultados mostrados en las siguientes imágenes:




Cam
[11.47, 1.51, 1.49, 8.72, 6.82]
196×219
Det...


Cam
[23.26, 12.34, 7.68, 2.19, 34.24]
196×219
Det...


Cam
[11.03, 3.52, 22.98, 3.94, 6.33]
196×219
Det...

4. Ejercicio “FILTROS”

4.1. Enunciado

Muestra el efecto de diferentes filtros sobre la imagen en vivo de la webcam. Selecciona con el teclado el filtro deseado y modifica sus posibles parámetros (p.ej. el nivel de suavizado) con las teclas o con trackbars. Aplica el filtro en un ROI para comparar el resultado con el resto de la imagen.

4.2. Resolución

Para este ejercicio se han considerado tres filtros; median, blur y box. El control de la selección entre los diferentes filtros se hace a través de las teclas que se muestran en la ayuda. Mediante variables booleanas se establece el filtro que ha sido seleccionado mediante teclado.

Este filtro seleccionado solamente se va a aplicar cuando se haya seleccionado una región de interés en la cámara en vivo. Los valores de los parámetros de los filtros son modificables mediante un trackbar que aparecen en pantalla durante la ejecución y existe un control de errores para que el valor seleccionado sea aceptable por cada filtro. Además, se aplica únicamente dentro de la región de interés y no fuera.

La dificultad del ejercicio radica en la transformación de los píxeles dentro de la ROI en píxeles alterados por los filtros calculados. Es posible que, si se consideraran muchos más filtros, el control de los mismos mediante variables booleanas se haga inmanejable. Es por ello que, si en un futuro se tuviera en cuenta este ejercicio para una aplicación seria con más funcionalidad, haría uso de un diccionario que asociara un filtro a una variable y que mediante una función se obtuviera el filtro a aplicar en cada momento. En este caso no me parece necesario ni pertinente por el contexto del propio ejercicio.

De nuevo, adjunto el código casi al completo porque todo me parece importante y forma parte del núcleo de la resolución:

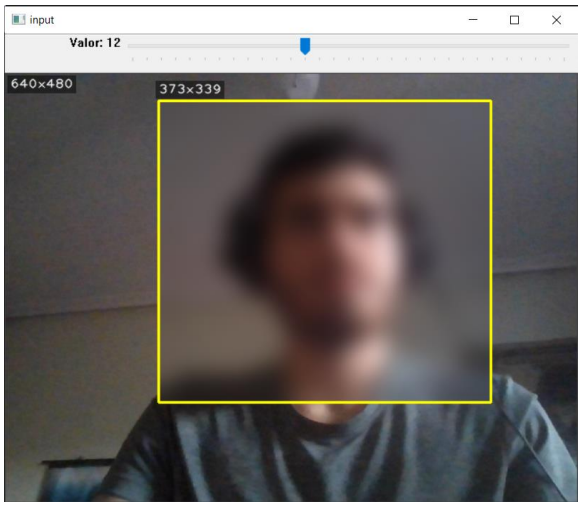
```
def nada(v):
    pass

# Se crea la ventana que mostrará los frames capturados
cv.namedWindow("input")
cv.createTrackbar("Valor", "input", 5, 30, nada)
cv.moveWindow('input', 0, 0)

region = ROI("input")
gblur = False
box = False
median = False
```

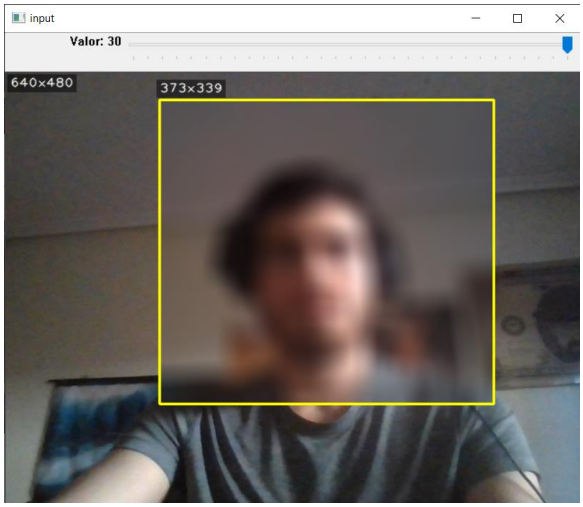
```
# Se itera sobre cada frame capturado
for key, frame in autoStream():

    help.show_if(key, ord('h'))

    if key == ord('g'):
        gblur = True
        box = False
        median = False
    elif key == ord('b'):
        box = True
        gblur = False
        median = False
    elif key == ord('m'):
        median = True
        box = False
        gblur = False

    # Si se ha seleccionado una región se guardan sus coordenadas
    if region.roi:
        [x1,y1,x2,y2] = region.roi
        filterValue = cv.getTrackbarPos('Valor','input')

        reg = frame[y1:y2+1, x1:x2+1]

        # Se aplica el filtro que esté seleccionado
        filtro = reg
        if gblur:
            filtro = cv.GaussianBlur(reg, (0,0), filterValue) if filterVa
lue > 0 else reg
            box = False
            median = False
        elif box:
            filtro = cv.boxFilter(reg, -
1, (filterValue, filterValue)) if filterValue > 0 else reg
            gblur = False
            median = False
        elif median:
            if filterValue > 0 and filterValue % 2 == 1:
                filtro = cv.medianBlur(reg, filterValue)
            elif filterValue > 0 and filterValue % 2 == 0:
                filtro = cv.medianBlur(reg, filterValue-1)
            else:
                filtro = reg
            gblur = False
            box = False
```

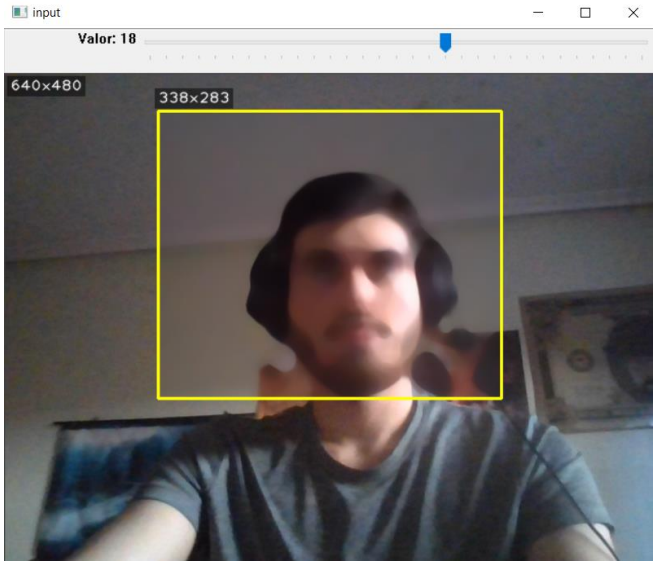
```
        # Se aplica el filtro sobre la ROI
        frame[y1:y2+1, x1:x2+1] = filtro

        # Si se pulsa 'c' se copia la ROI a otra ventana
        if key == ord('c'):
            trozo = frame[y1:y2+1, x1:x2+1]
            cv.imshow("trozo", trozo)

        # Si se pulsa 'x' se deselecciona la ROI
        if key == ord('x'):
            region.roi = []

        # Se dibuja un rectángulo en las coordenadas de la ROI mostrando
su tamaño
        cv.rectangle(frame, (x1,y1), (x2,y2), color=(0,255,255), thicknes
s=2)
        putText(frame, f'{x2-x1+1}x{y2-y1+1}', orig=(x1,y1-8))

    # Se dibuja cada frame captado por la webcam
    h,w,_ = frame.shape
    putText(frame, f'{w}x{h}')
    cv.imshow('input',frame)
```

4.3. Ejemplos

Como ejemplos muestro la aplicación de cada uno de los tres filtros que he tenido en cuenta, la primera imagen a la izquierda muestra un filtro gaussiano, el de la derecha es un filtro ‘box’ y el correspondiente a la imagen inferior es un filtro de mediana.






input
Valor: 18
640×480
338×283

5. Ejercicio “SIFT”

5.1. Enunciado

Escribe una aplicación de reconocimiento de objetos (p. ej. carátulas de CD, portadas de libros, cuadros de pintores, etc.) con la webcam basada en el número de coincidencias de *keypoints*.

5.2. Resolución

Este ejercicio es parecido a “COLOR” en el sentido de que sigo la misma filosofía de tener una lista de modelos que puedo añadir mediante teclado, habiéndoles calculado previamente los *keypoints* utilizando la función ‘detectAndCompute’ de OpenCV.

Estos modelos son comparados con el *frame* que esté siendo proyectado en cámara, y el modelo que más coincidencias presente con respecto al número total de *keypoints* genera la tasa de acierto más alta y será seleccionado para mostrarlo al lado del *streaming* de la cámara. La segunda mejor tasa de acierto se tiene en cuenta también para mostrarla y comprobar que se está tomando el modelo que más se asemeja en cada momento.

Lo explicado junto a pequeños detalles adicionales pueden comprobarse en el código que muestro a continuación:

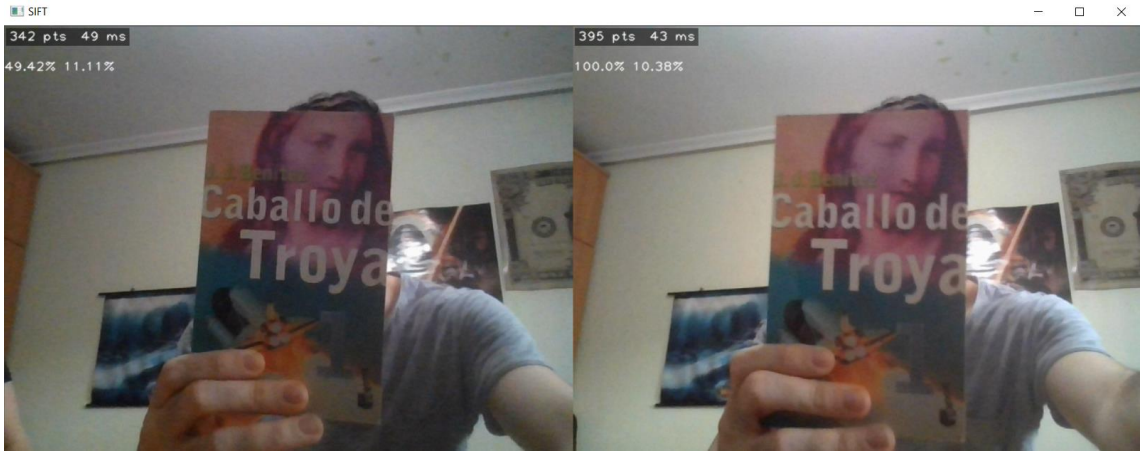
```
sift = cv.AKAZE_create()
matcher = cv.BFMatcher()
modelos = []

for key, frame in autoStream():

    t0 = time.time()
    keypoints, descriptors = sift.detectAndCompute(frame, mask=None)
    if len(keypoints) == 0:
        descriptors = None

    t1 = time.time()
    putText(frame, f'{len(keypoints)} pts  {1000*(t1-t0):.0f} ms')

    if key == ord('c'):
        modelos.append((keypoints, descriptors, frame))

    if len(modelos) == 0:
        flag = cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS
        cv.drawKeypoints(frame, keypoints, frame, color=(100,150,255), fl
ags=flag)
        cv.imshow('SIFT', frame)
```

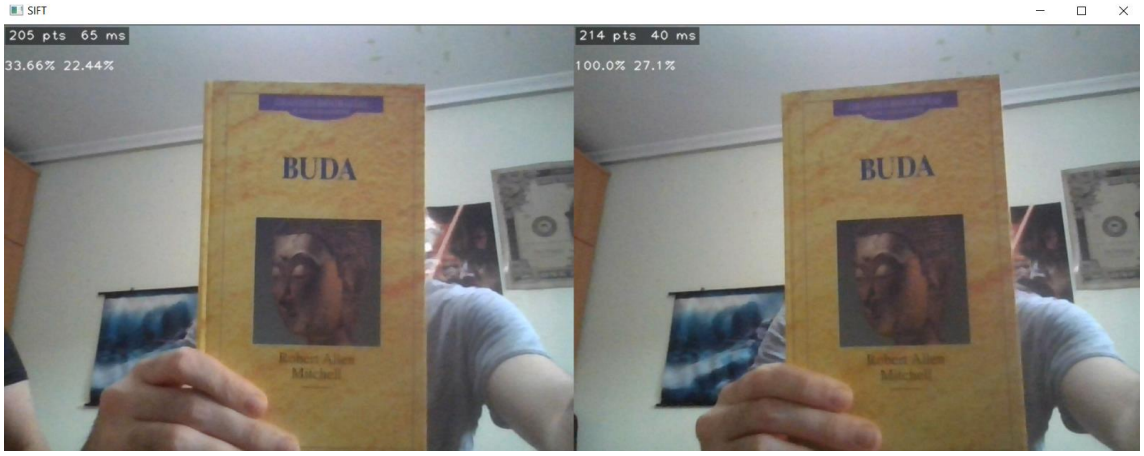
```
    else:
        mejorPorcentaje = 0
        segundoMejorPorcentaje = 0
        mejorFrame = frame
        t2, t3 = 0, 0
        for modelo in modelos:

            t2 = time.time()
            # Se guardan las dos mejores coincidencias de cada punto
            matches = matcher.knnMatch(descriptors, modelo[1], k=2)
            t3 = time.time()

            # Se guardan las coincidencias que son mucho mejores que
            # que la segunda mejor. Si un punto se parece a dos puntos di
ferentes del modelo se elimina
            good = []
            for m in matches:
                if len(m) >= 2:
                    best,second = m
                    if best.distance < 0.75*second.distance:
                        good.append(best)

            porcentaje = len(good) * 100 / len(keypoints)
            if mejorPorcentaje < porcentaje:
                segundoMejorPorcentaje = mejorPorcentaje
                mejorPorcentaje = porcentaje
                mejorFrame = modelo[2]
            elif porcentaje > segundoMejorPorcentaje:
                segundoMejorPorcentaje = porcentaje


        putText(frame, f'{round(mejorPorcentaje,2)}% {round(segundoMejorP
orcentaje,2)}%',orig=(0,50))
        cv.imshow("SIFT",np.hstack([frame,mejorFrame]))
```

5.3. Ejemplos

Para comprobar el funcionamiento del programa, he utilizado los mismos objetos que en el ejercicio de ‘COLOR’. La diferencia en cuanto a interfaz es que ya no se muestra una lista con los modelos guardados, sino que se muestra en la derecha el modelo que más se parece según sus *keypoints* a lo que se muestra por cámara en cada momento. Además, se muestra en la esquina superior izquierda el porcentaje de acierto explicado en el apartado anterior, del mejor modelo y del segundo mejor. El resultado se puede comprobar en las imágenes que se muestran a continuación:


SIFT
198 pts 58 ms
31.82% 1.52%
195 pts 43 ms
100.0% 1.54%
WORLD
WARCRAFT


SIFT
342 pts 49 ms
49.42% 11.11%
395 pts 43 ms
100.0% 10.38%
Caballo de
Troya


SIFT
205 pts 65 ms
33.66% 22.44%
214 pts 40 ms
100.0% 27.1%
BUDA


SIFT
320 pts 55 ms
13.12% 4.38%
357 pts 43 ms
100.0% 0%
PS4
Fallout 4


SIFT
136 pts 44 ms
16.18% 0.74%
109 pts 44 ms
100.0% 0.92%

6. Ejercicio “RECTIF”

6.1. Enunciado

Rectifica la imagen de un plano para medir distancias (tomando manualmente referencias conocidas). Por ejemplo, mide la distancia entre las monedas en `coins.png` o la distancia a la que se realiza el disparo en `goleder.png`. Verifica los resultados con imágenes originales tomadas por ti.

6.2. Resolución

En este ejercicio, lo primero que he hecho ha sido idear una función para conseguir y marcar los cuatro puntos necesarios para poder rectificar un el plano de la imagen. Cuando se consiguen, se utilizan para obtener la matriz H de la homografía y aplicarla dentro de la función ‘warpPerspective()’ para rectificar el plano.

Esta imagen rectificada aparece en otra ventana, en la que se pueden volver a marcar dos puntos, esta vez para medir la distancia entre objetos de la imagen. Esta distancia es la distancia euclídea entre puntos, convertida a centímetros, que va en función de las proporciones del paralelogramo marcado anteriormente y su tamaño real una vez rectificado. Para ejecutarlo es necesario determinar la ruta de la imagen como único parámetro de entrada. El código más relevante de este ejercicio puede consultarse como a continuación como viene siendo habitual en el documento:

```
# Variable para guardar las coordenadas de las esquinas del carnet
esquinasRect = []

# Control del bucle principal
puntosMarcados = False

# Callback que va guardándose las coordenadas de los clicks mientras mues
tra los puntos

def getCoordsRect(event, x, y, flags, param):

    # Si se hace click izquierdo
    if event == cv.EVENT_LBUTTONDOWN:

        # Se añade la posición del click a la lista de esquinas
        esquinasRect.append([x,y])

        # Mientras no se tengan 4 puntos se pueden ir agregando y uniéndo
se mediante una línea
        nEsquinas = len(esquinasRect)

        if nEsquinas < 4:
```

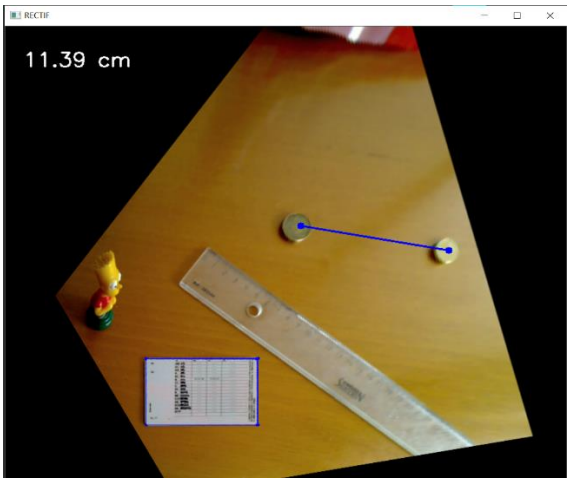
```
            # Se dibuja el círculo donde se hace click
            cv.circle(imgToRectif, (x, y), 4, (255,0,0), -1)

            cv.line(imgToRectif, (esquinasRect[nEsquinas-
2][0], esquinasRect[nEsquinas-2][1]),
                                 (esquinasRect[nEsquinas-
1][0], esquinasRect[nEsquinas-1][1]),
                                 (255,0,0), 2)

        # Si se tienen 4 ya hay que cerrar el rectángulo
        elif nEsquinas == 4:

            # Se dibuja el círculo donde se hace click
            cv.circle(imgToRectif, (x, y), 4, (255,0,0), -1)

            cv.line(imgToRectif, (esquinasRect[3][0], esquinasRect[3][1])
,
                                 (esquinasRect[0][0], esquinasRect[0][1])
,
                                 (255,0,0), 2)
            cv.line(imgToRectif, (esquinasRect[2][0], esquinasRect[2][1])
,
                                 (esquinasRect[3][0], esquinasRect[3][1])
,
                                 (255,0,0), 2)
            global puntosMarcados
            puntosMarcados = True

        # Cuando se hace click más de 4 veces se resetea la lista de esqu
inas
        else:
            esquinasRect.clear()

# Variable para guardar las coordenadas de los puntos entre los que se qu
iere medir una distancia
puntos = []


# Callback que se utiliza para medir y mostrar la distancia entre dos pun
tos de la imagen rectificada

def getDistanciaPuntos(event, x, y, flags, param):

    # Si se hace click izquierdo
    if event == cv.EVENT_LBUTTONDOWN:

        puntos.append([x, y])

        # Si no hay dos puntos ya, se dibuja en ella un círculo
```

```
        if len(puntos) < 2:
            cv.circle(imgRectif, (x, y), 5, (255,0,0), -1)

        # Si ya se tienen dos puntos, se dibuja el segundo y se unen con
una línea
        elif len(puntos) == 2:
            cv.circle(imgRectif, (x, y), 5, (255,0,0), -1)
            cv.line(imgRectif, (puntos[0][0], puntos[0][1]), (puntos[1][0
], puntos[1][1]), (255,0,0), 2)

            # Cálculo de la distancia euclídea
            dx2 = (puntos[0][0]-puntos[1][0])**2
            dy2 = (puntos[0][1]-puntos[0][1])**2
            distancia = math.sqrt(dx2 + dy2)

            # Se realiza una conversión a centímetros
            distanciaReal = round(distancia * h / longitud, 2)
            cv.putText(imgRectif, repr(distanciaReal) + " cm", org=(25, 6
0), fontFace=0, fontScale=1, color=(255,255,255), thickness=2)


# Medidas del carnet de conducir en cm
#w, h = 8.6, 5.4

# Medidas del paralelogramo que se va a medir en el área del campo
w, h = 1650, 1100
aspectRatio = w/h

# Longitud del cuadro rectificado
longitud = 100

# Posiciones en la imagen del cuadro una vez rectificado
x,y = 200,500
real = np.array([[x,y],
                 [x+longitud*aspectRatio, y],
                 [x+longitud*aspectRatio, (y+longitud)],
                 [x, y+longitud]])

# Se lee la imagen de entrada a rectificar
imgToRectif = cv.imread(sys.argv[1])

# Control del bucle
isRectificada = False

while True:

    # Presionando 'Esc' nos salimos
    key = cv.waitKey(1) & 0xFF
    if key == 27: break
```

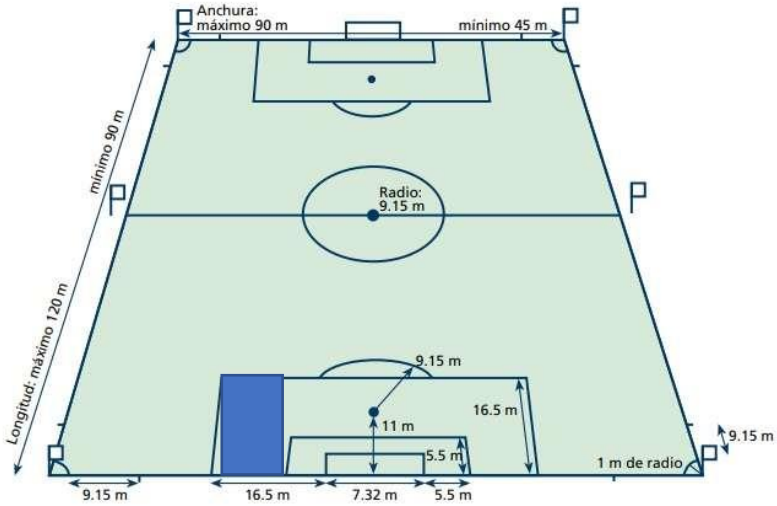
```
    # Se muestra la imagen para marcar los puntos del objeto con medidas
conocidas
    cv.setMouseCallback("ImgToRectif", getCoordsRect)
    cv.imshow("ImgToRectif", imgToRectif)

    # Si se han marcado
    if not isRectificada and puntosMarcados:
        view = np.array([
            [esquinasRect[0][0], esquinasRect[0][1]],
            [esquinasRect[1][0], esquinasRect[1][1]],
            [esquinasRect[2][0], esquinasRect[2][1]],
            [esquinasRect[3][0], esquinasRect[3][1]]])

        H, _ = cv.findHomography(view, real)
        # Se copia la imagen sin dibujos primero por si se quiere volver
a medir
        # Ref: https://stackoverflow.com/questions/47005416/opencv-clear-
screen
        cleanImgRectif = cv.warpPerspective(imgToRectif, H, (800, 800))
        imgRectif = cleanImgRectif.copy()
        isRectificada = True

    # Si ya se tienen los puntos sobre el objeto conocido se dibuja la im
agen con rectificación de plano
    elif isRectificada:
        cv.setMouseCallback("RECTIF", getDistanciaPuntos)
        cv.imshow("RECTIF", imgRectif)

        # Si se quiere volver a medir, pulsar 'r'
        if key == ord('r'):
            imgRectif = cleanImgRectif.copy()
            puntos.clear()

cv.destroyAllWindows()
```

6.3. Ejemplos

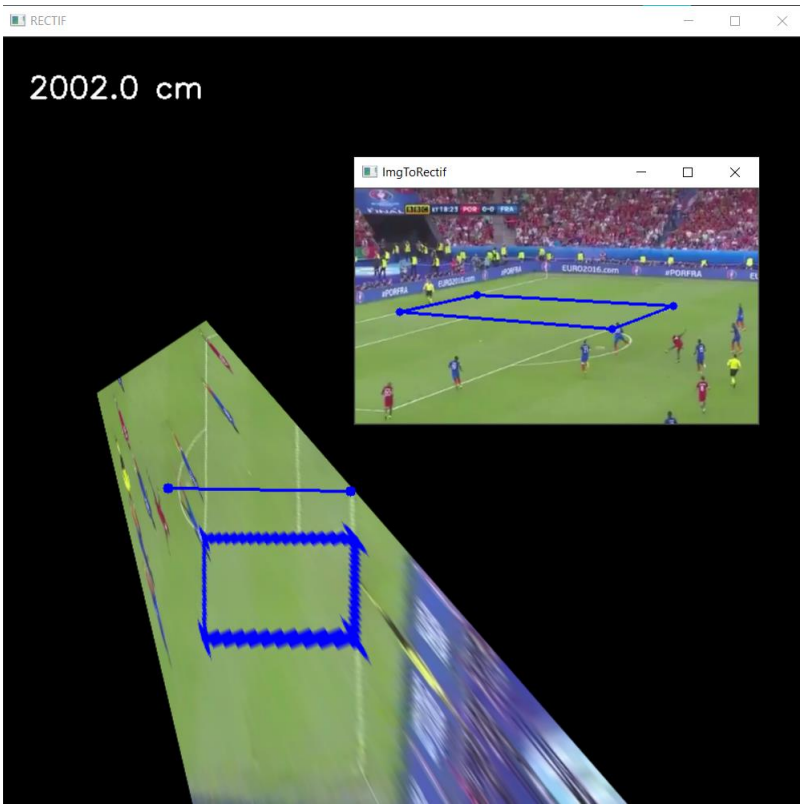
Para la primera imagen muestro las dos ventanas que se generan; la primera para marcar los puntos, y la segunda con la imagen rectificada. Según mi programa, la distancia que hay entre ambas monedas es de aproximadamente **11.39 cm**.



Para la imagen del gol de Eder, he considerado el rectángulo dentro del área del campo marcado en azul, que tiene unas medidas de 16.5 m de ancho y 11 m de alto.

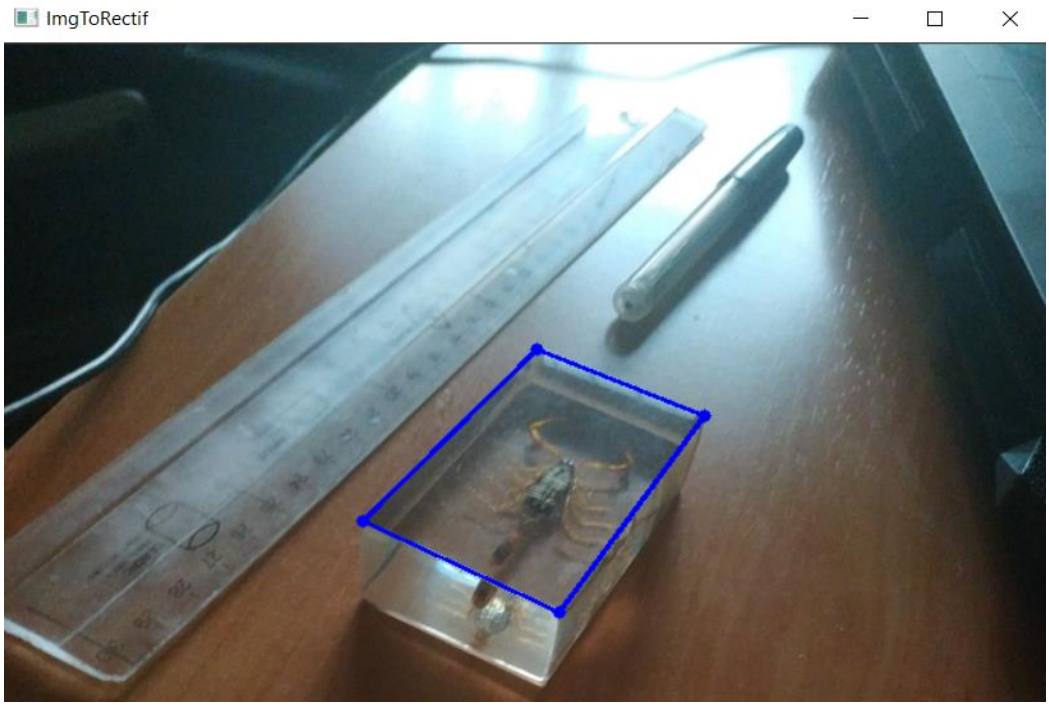


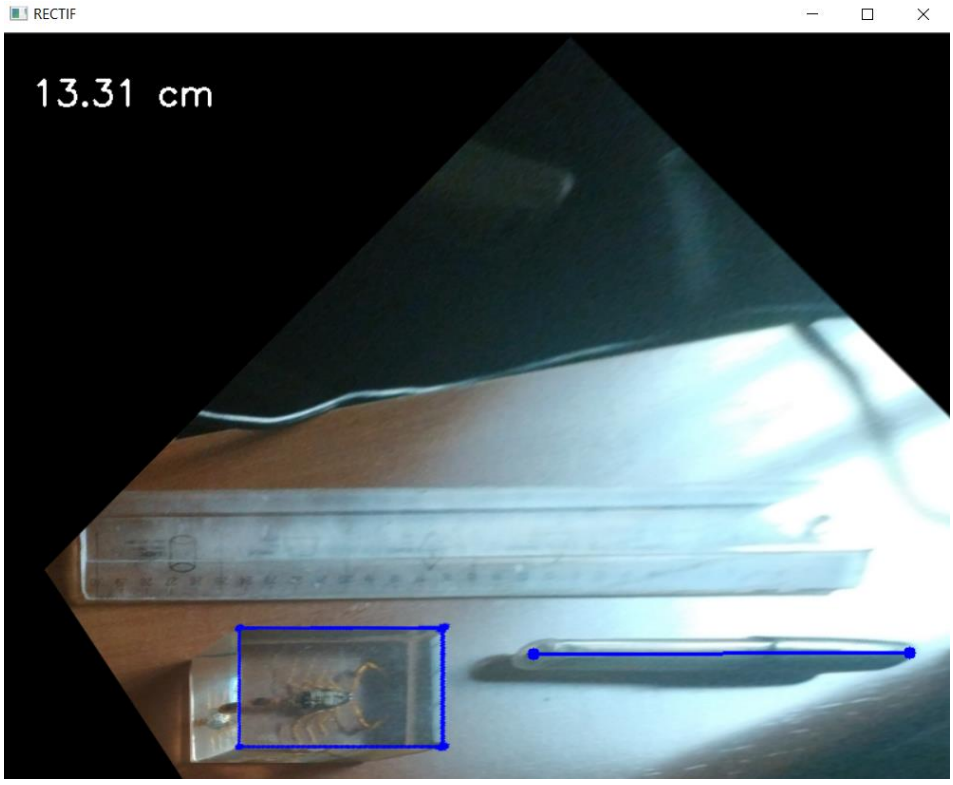
La distancia medida desde la posición del disparo es de unos **20 m**. aproximadamente como se puede ver en la siguiente imagen:



Para el ejemplo propio, me he propuesto medir la longitud de mi bolígrafo favorito utilizando el programa desarrollado. El bolígrafo, como puede verse, mide unos 13.2 cm. aproximadamente:

Con la rectificación de una imagen tomada con ángulo sobre el bolígrafo y mi referencia (un escorpión dentro de un bloque de resina), se obtiene una medida aproximada a la real, de **13.31 cm**:


ImgToRectif


RECTIF
13.31 cm

7. Ejercicio “PANO”

7.1. Enunciado

Crea automáticamente un mosaico a partir de las imágenes en una carpeta. Las imágenes no tienen por qué estar ordenadas ni formar una cadena lineal y no sabemos el espacio que ocupa el resultado. El usuario debe intervenir lo menos posible. Recuerda que debe tratarse de una escena plana o de una escena cualquiera vista desde el mismo centro de proyección. Debes usar homografías. Compara el resultado con el que obtiene la utilidad de stitching de OpenCV.

7.2. Resolución

La resolución de este ejercicio sería mucho más sencilla sin la restricción de que no tienen por qué estar en orden las imágenes, ya que bastaría con ir uniendo con la siguiente y combinándolas usando homografías y alterando el plano.

El código está basado en ‘notebooks’ de clase y en la experiencia obtenida durante el desarrollo del ejercicio ‘SIFT’. La idea es que se parte de la primera imagen que llega, y se itera sobre el resto. La imagen que más puntos comparta (o que mejor ratio de semejanza tenga) con la primera es la que se combina finalmente con ella, rectificando el plano mediante ‘warpPerspective’ durante su combinación. Es a partir de esta imagen combinada donde se busca nuevamente otra imagen siguiendo el mismo proceso.

Es importante que la imagen combinada se inserte al principio de la lista que contiene las imágenes para que funcione correctamente y borrar las que han sido combinadas para no volver a tenerlas en cuenta, como se ve en el método ‘main()’. Para ejecutarlo, es necesario establecer la ruta donde se encuentran las imágenes e incluir todos los archivos como único parámetro de entrada (utilizando * por ejemplo). Los detalles se pueden consultar en el código:

```
import cv2 as cv
import numpy as np
import sys
import matplotlib.pyplot as plt
from glob import glob


def getKeypointsAndDescriptors(readImgs):
    kpsAndDes = []
    for img in readImgs:
        kps, des = sift.detectAndCompute(img, None)
        kpsAndDes.append((img, kps, des))
    return kpsAndDes
```

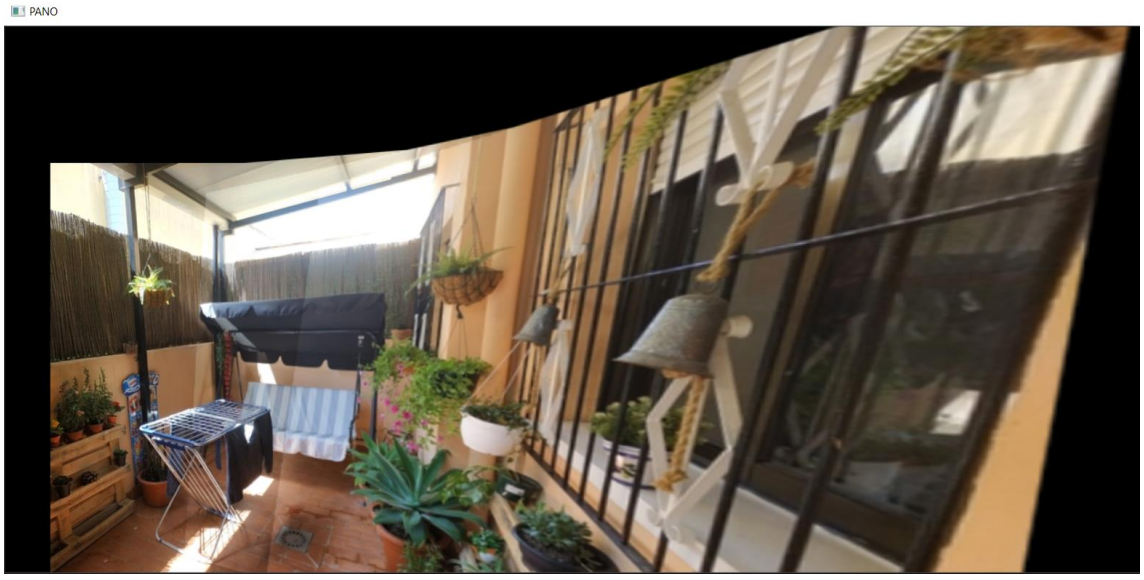
```
def findCandidate(kpsAndDes, kad):

    bestRatio, i = 0, 0
    bestGood, bestImg, bestKad = None, None, None

    for kadi in kpsAndDes:
        ratio,good = bestMatch(kad[2], kadi[2], kad[1])

        if ratio > bestRatio:
            bestRatio = ratio
            bestGood = good
            bestKad = kadi
            i += 1

    return (bestKad, bestGood, i)


def bestMatch(des1, des2, kps1):
    good = []
    matches = matcher.knnMatch(des2, des1, k=2)

    for m in matches:
        if len(m) == 2:
            best,second = m
            if best.distance < 0.75 * second.distance:
                good.append(best)

    ratio = len(good) / len(kps1)
    return (ratio,good)


def desp(desp):
    dx,dy = desp
    return np.array([[1,0,dx],
                     [0,1,dy],
                     [0,0,1]])


def combineImages(img1, img2, H):
    return np.maximum(cv.warpPerspective(img2, desp((50,150)) @ np.eye(3)
,(1800,600)),
                      cv.warpPerspective(img1, desp((50,150)) @ H,(1800,6
00)))


def readImages(inputImgs):
    readImgs = []
```

```
    for i in inputImgs:
        img = cv.imread(i)
        readImgs.append(img)

    return readImgs


sift = cv.AKAZE_create()
matcher = cv.BFMatcher()

def main():

    inputImgs = glob(sys.argv[1])
    readImgs = readImages(inputImgs)
    newImg = None

    kpsAndDes = getKeypointsAndDescriptors(readImgs)

    while len(kpsAndDes) != 1:

        kad = kpsAndDes[0]
        (bestKad, good, imgIndex) = findCandidate(kpsAndDes[1:], kad)

        src_pts = np.array([kad[1][m.trainIdx].pt for m in good]).astype(
np.float32).reshape(-1,2)
        dst_pts = np.array([bestKad[1][m.queryIdx].pt for m in good]).ast
ype(np.float32).reshape(-1,2)
        H, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC, 3)

        newImg = combineImages(kad[0], bestKad[0], H)

        kpsAndDes.pop(imgIndex)
        kpsAndDes = kpsAndDes[1:]
        k,d = sift.detectAndCompute(newImg, None)
        kpsAndDes.insert(0,(newImg, k, d))


    while(True):
        key = cv.waitKey(1) & 0xFF
        if key == 27: break
        cv.imshow('PANO', kpsAndDes[0][0])

main()
```

7.3. Ejemplos

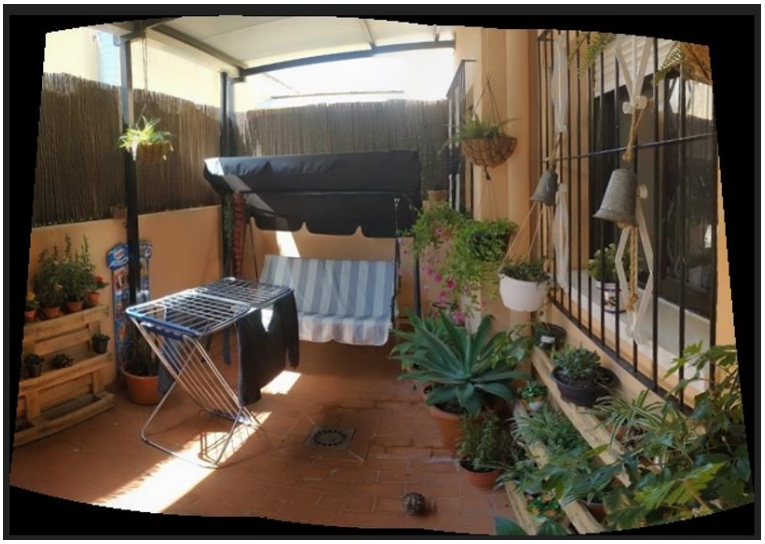
El primer ejemplo sobre el que he aplicado el código es con las imágenes de la facultad, obteniendo el siguiente resultado:

Para el ejemplo propio he tomado imágenes en mi casa, fijando la mano y girando el propio móvil para tomarlas desde el mismo centro. El resultado es el siguiente:



El resultado obtenido con “stitcher.py” es mucho mejor, las distorsiones producidas no se aprecian tanto y los bordes de las imágenes están recortados y se obtiene un mejor acabado. Aún así, no logra coser todas las imágenes. Mi teoría es que el funcionamiento en Windows es distinto, debido a que para poder ejecutar el ‘sift’ hay que utilizar la siguiente línea “cv.AKAZE_create()”, lo que parece empeorar el rendimiento con la panorámica de la facultad:

Con la imagen propia parece funcionar correctamente:



8. Ejercicio “RA”

8.1. Enunciado

Crea un efecto de realidad aumentada **interactivo**: esto significa que a) los objetos virtuales deben cambiar de forma, posición o tamaño siguiendo alguna lógica; b) el usuario puede observar la escena cambiante desde cualquier punto de vista moviendo la cámara alrededor del marcador; y c) el usuario puede marcar con el ratón en la imagen puntos del plano de la escena para interactuar con los objetos virtuales.

8.2. Resolución

Para este ejercicio se parte de uno de los scripts de poses disponible en el ‘github’ de la asignatura. La primera característica relevante es que he creado formas nuevas aparte del cubo disponible con la librería. El segundo detalle es que la forma es capaz de cambiar de tamaño y alterar su figura mediante teclado (pulsando ‘h’ se pueden consultar qué teclas están involucradas).

Además, he hecho que la figura cambie de color en función el color del píxel donde se haya hecho click. Esta interacción es muy simple y es por ello que se ha implementado además una detección de click dentro de la figura con la función ‘dentroContornos()’, que exige el uso de la matriz de cámara para la transformación entre coordenadas 3D y 2D, calculada mediante la función ‘htrans()’. Así, cuando se detecta un click dentro del espacio que ocupa la figura, esta comienza a moverse (con el marcador quieto) describiendo un giro dado por las aristas del marcador del folio de forma ininterrumpida hasta que se le vuelve a hacer click, quedándose entonces parada hasta el siguiente click, etc.

El trozo de código que adjunto a continuación obvia las funciones que venían dadas por el script de partida y contiene aquellas partes interesantes que se corresponden con lo explicado:

```
# Callback para detectar clicks
def fun(event, x, y, flags, param):
    global puntoClick
    if event == cv.EVENT_LBUTTONDOWN:
        puntoClick = (y,x)


# Ref: https://stackoverflow.com/questions/36399381/whats-the-fastest-
way-of-checking-if-a-point-is-inside-a-polygon-in-python
def dentro_contornos(contornos, punto):
    x = punto[0]
    y = punto[1]
```

```
    contornos = contornos[0]
    n = len(contornos)
    inside = False
    p2x = 0.0
    p2y = 0.0
    xints = 0.0
    p1x,p1y = contornos[0]
    for i in range(n+1):
        p2x,p2y = contornos[i % n]
        if y > min(p1y,p2y):
            if y <= max(p1y,p2y):
                if x <= max(p1x,p2x):
                    if p1y != p2y:
                        xints = (y-p1y)*(p2x-p1x)/(p2y-p1y)+p1x
                    if p1x == p2x or x <= xints:
                        inside = not inside
        p1x,p1y = p2x,p2y

    return inside

# Figura pirámide (propia)
pyramid = np.array([[0, 0, 1],
                    [1, 0, 1],
                    [1, 1, 1],
                    [0, 1, 1],
                    [0, 0, 1],
                    [0.5, 0.5, 1.5],
                    [1, 0, 1],
                    [0.5, 0.5, 1.5],
                    [1, 1, 1],
                    [0.5, 0.5, 1.5],
                    [0, 1, 1],
                    [0.5, 0.5, 1.5]])

# Direccion en la que se mueve el muñeco

for key,frame in stream:
    key = cv.waitKey(1) & 0xFF
    if key == 27: break

    # Si se detecta click se activa o desactiva la animación
    if puntoClick != None:
        ant = clickFigura
        clickFigura = dentro_contornos([htrans(M,formaActual/2 * factorTa
m + [0,mov,0]).astype(int) for M in poses], (puntoClick[1],puntoClick[0])
)
        if ant == True: clickFigura = not clickFigura
        puntoClick = None
```

```
    # La pirámide realiza un desplazamiento en función de las aristas del
 marcador si se ha hecho click encima
    if clickFigura:
        if nframes % 10 == 0:
            pyramid += marker[indiceMarker]
            indiceMarker += 1

        if indiceMarker == len(marker):
            pyramid = pyramidOrg.copy()
            indiceMarker = 0
        nframes += 1


    # Control de la figura mediante teclado
    if key == ord('c') and factorTam < 4.5: # Aumentar tam
        factorTam *= 1.1
    elif key == ord('d') and factorTam > 0.5: # Disminuir tam
        factorTam /= 1.1
    elif key == ord('m'): # Cambiar la forma de la figura
        if not cubo:
            formaActual = cube
            cubo = True
        else:
            formaActual = pyramidInv
            cubo = False

    g = cv.cvtColor(frame,cv.COLOR_BGR2GRAY)
    cs = extractContours(g, minarea=5, reduprec=2)

    good = polygons(cs,6,3)

    poses = []
    for g in good:
        pose = bestPose(K,g,marker)
        if pose.rms < 2:
            poses += [pose.M]

    # Se dibujan las figuras en función del control
    cv.drawContours(frame,[htrans(M,marker).astype(int) for M in poses],
-1, (0,255,255), 1, lineType)
    cv.drawContours(frame,[htrans(M,pyramid/2 * factorTam + [0,mov,0]).as
type(int) for M in poses], -1, colRGB, 3, lineType)
    cv.drawContours(frame, [htrans(M,formaActual/2 * factorTam + [0,mov,0
]).astype(int) for M in poses], -1, colRGB, 3, lineType)

    cv.imshow('Realidad Aumentada',frame)
```

8.3. Ejemplos

Para este ejercicio, he realizado un vídeo donde se puede ver cómo se puede cambiar de forma y mover la figura superior que dibujo haciendo click sobre ella. El vídeo se ha tomado con OBS y el puntero no se mueve acorde a la realidad, pero puede ser comprobado ejecutando el código y viendo que funciona.

9. Ejercicio “ANON”

9.1. Enunciado

Modifica el ejemplo de reconocimiento de caras `DL/facerec` para seleccionar caras pinchando con el ratón (o tomándolas de un directorio) para que cuando se reconozcan en las imágenes se oculten (emborronándolas o pixelizándolas).

9.2. Resolución

En este ejercicio se parte de un script base, del cual se adapta la función “pixelate_face”, que recibe la ROI donde se encuentra la cara detectada, y se divide en pequeños bloques que calculan el valor medio de color que tiene y se aplica a todo el bloque este valor medio, de forma que cada bloque toma un valor fijo logrando el efecto de pixelado sobre la ROI. Este pixelado solamente se aplica si la cara detectada a través de la cámara hace ‘match’ con alguna de las caras detectada en una carpeta con fotos de gente cuando se comparan internamente.

El funcionamiento del código para esto último es sencillo; primero, se detectan las caras en las imágenes, después se detectan caras a través de la cámara y se realiza un bucle para buscar una coincidencia de entre todas las imágenes. Si la coincidencia se da, se invoca a “pixelate_face”, y en caso contrario no se aplica el pixelado.

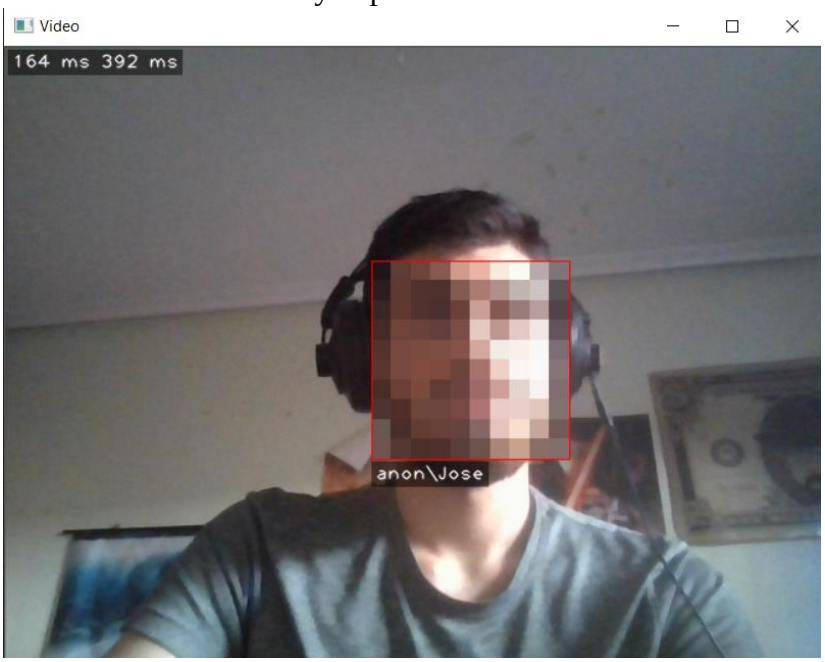
```
#!/usr/bin/env python

# Se parte del script sugerido en el enunciado

import face_recognition
import cv2 as cv
import numpy as np
import time
from umucv.util import putText
from umucv.stream import autoStream
import glob


def readrgb(filename):
    return cv.cvtColor(cv.imread(filename), cv.COLOR_BGR2RGB)


def readModels(path):
    fmods = sorted([name for name in glob.glob(path+'/*.*') if name[-
3:] != 'txt'])
    models = [ readrgb(f) for f in fmods ]
    return fmods, models
```

```
# Adaptado de: https://www.pyimagesearch.com/2020/04/06/blur-and-
anonymize-faces-with-opencv-and-python/
def pixelate_face(face_frame, blocks=3):

    # Se divide el frame en los bloques que se quieran
    (h, w) = face_frame.shape[:2]
    xCoords = np.linspace(0, w, blocks + 1, dtype="int")
    yCoords = np.linspace(0, h, blocks + 1, dtype="int")

    # Se recorren todos los bloques en los que ha dividido el frame
    for i in range(1, len(yCoords)):
        for j in range(1, len(xCoords)):

            # Se toman las coordenadas del bloque actual
            x1 = xCoords[j - 1]
            y1 = yCoords[i - 1]
            x2 = xCoords[j]
            y2 = yCoords[i]

            # Se extrae la región de interés asociada al bloque actual
            roi = face_frame[y1:y2, x1:x2]

            # Se calcula el valor medio de esa región
            (B, G, R) = [int(x) for x in cv.mean(roi)[:3]]

            # Se establece ese valor medio en el bloque actual
            cv.rectangle(face_frame, (x1, y1), (x2, y2), (B, G, R), -1)

    return face_frame

filenames, models = readModels('anon')
names = [ x.split('/')[-1].split('.')[0].split('-
')[0] for x in filenames ]
encodings = [ face_recognition.face_encodings(x)[0] for x in models ]

print(encodings[0].shape)

for key, frame in autoStream():

    t0 = time.time()

    face_locations = face_recognition.face_locations(frame)
    t1 = time.time()

    face_encodings = face_recognition.face_encodings(frame, face_location
s)
    t2 = time.time()
```

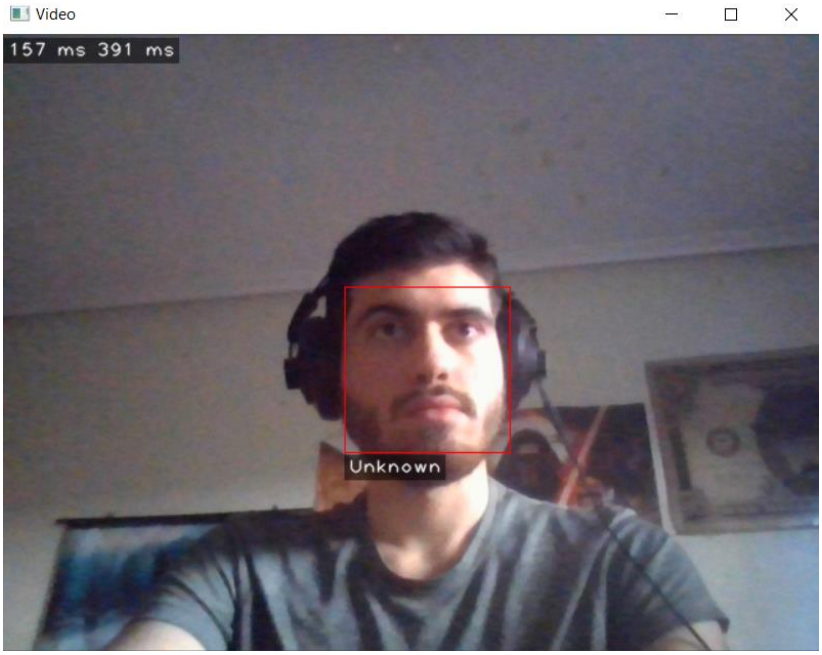
```
    for (top, right, bottom, left), face_encoding in zip(face_locations,
face_encodings):
        match = face_recognition.compare_faces( encodings, face_encoding)

        name = "Unknown"
        for n, m in zip(names, match):
            if m:
                name = n

        # El código pixela la cara si se encuentra en una imagen del dire
ctorio /gente
        if name != "Unknown":
            frame[top:bottom,left:right] = pixelate_face(frame[top:bottom
,left:right], 10)

        cv.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 1)
        putText(frame, name, orig=(left+3,bottom+16))

    putText(frame, f'{(t1-t0)*1000:.0f} ms {(t2-t1)*1000:.0f} ms')

    cv.imshow('Video', frame)
```

9.3. Ejemplos

Como ejemplo, aparezco yo en la cámara tras añadir una imagen mía a la carpeta donde se toman todas las imágenes. El programa, al detectarme a mí en pantalla, selecciona la ROI donde ha sido detectada la cara y la pixela:



Si retiro mi imagen del directorio, no se detecta mi cara en la lista de imágenes y no se me pixela como se ve a continuación:



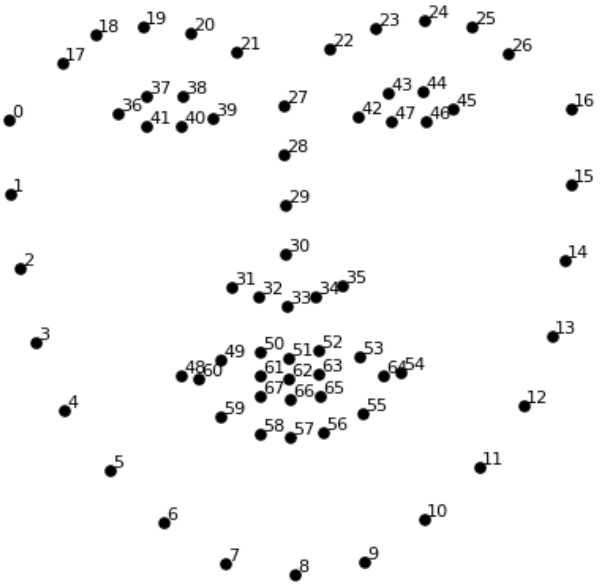
10. Ejercicio “DLIB”

10.1. Enunciado

Escribe una aplicación de tu invención que utilice los marcadores de cara obtenidos por el *shape detector* disponible en dlib.

10.2. Resolución

La aplicación desarrollada para este ejercicio utiliza los *landmarks* de la cara para colocar en la posición de los ojos una imagen en tiempo real , en mi caso, el signo del dólar. Los puntos que se utilizan para detectar caras en el predictor ‘shape_predictor_68_face_landmarks’ se pueden ver en la siguiente figura:



La idea del código se basa en obtener esos *landmarks* y acceder a los puntos de interés de la cara, en este caso los que van del 36 al 47, para tomar sus posiciones y colocar de la forma más centrada posible la imagen encima. El tamaño de la imagen tuvo que ser alterado para que tuviera una proporción razonable.

Una vez obtenidas las posiciones y los centros necesarios la dificultad radica en alterar el área donde se hayan los ojos dentro de cada frame para dibujar el dólar encima. El resultado es un filtro parecido al de aplicaciones como Instagram o Snapchat, aunque original ya que no me consta su existencia a día de hoy.

Para lanzarlo en suficiente con tener el script y la carpeta ‘dlib’ en el mismo directorio. El código se muestra a continuación para su consulta como en los ejercicios anteriores:

```
import dlib
import cv2 as cv
```

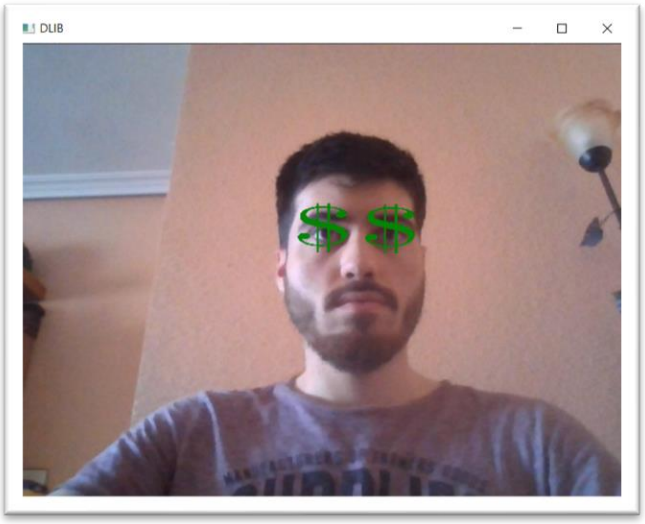
```
import numpy as np
from math import hypot
from umucv.stream import autoStream

# Código inspirado en: https://pysource.com/2019/03/25/pigs-nose-
instagram-face-filter-opencv-with-python/

# Función que devuelve las posiciones relevantes del ojo izq dados unos l
andmarks

def getPosOjoIzq(landmarks):
    eyeCenter = (landmarks.part(36).x + 10, landmarks.part(40).y - 10)
    eyeLeft = (landmarks.part(36).x, landmarks.part(36).y)
    eyeRight = (landmarks.part(39).x, landmarks.part(39).y)

    return (eyeCenter, eyeLeft, eyeRight)


# Función que devuelve las posiciones relevantes del ojo der dados unos l
andmarks

def getPosOjoDer(landmarks):
    eyeCenter = (landmarks.part(42).x + 15, landmarks.part(45).y - 5)
    eyeLeft = (landmarks.part(42).x, landmarks.part(42).y)
    eyeRight = (landmarks.part(45).x, landmarks.part(45).y)

    return (eyeCenter, eyeLeft, eyeRight)


# Función que devuelve el área ocupada por un ojo

def getEyeArea(top_left, eyeHeight, eyeWidth, eyeMask):
    eyeArea = frame[top_left[1]: top_left[1] + eyeHeight,
                    top_left[0]: top_left[0] + eyeWidth]

    return cv.bitwise_and(eyeArea, eyeArea, mask=eyeMask)


# Se prepara la máscara con el número de filas y columnas del frame
stream = autoStream()
_, frame = next(stream)
rows, cols, _ = frame.shape
eyeMask = np.zeros((rows, cols), np.uint8)

# Se carga el detector de caras
# El predictor se puede descargar en
# https://github.com/italojs/facial-landmarks-
recognition/blob/master/shape_predictor_68_face_landmarks.dat
detector = dlib.get_frontal_face_detector()
```

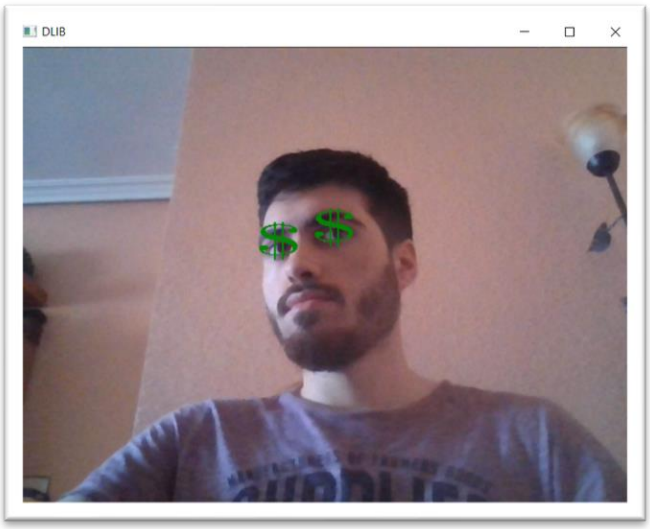
```
predictor = dlib.shape_predictor("dlib\\shape_predictor_68_face_landmarks
.dat")

eye_image = cv.imread("dlib\\dollar.png")

for key, frame in stream:

    eyeMask.fill(0)
    gray_frame = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)

    # El detector obtiene las caras en la cámara y se itera sobre ellas
    faces = detector(frame)
    for face in faces:

        # Se obtienen los 68 landmarks de la cara
        landmarks = predictor(gray_frame, face)

        # Se obtienen las posiciones relevantes del ojo en función de los
 landmarks
        (centerEye1, leftEye1, rightEye1) = getPosOjoIzq(landmarks)
        (centerEye2, leftEye2, rightEye2) = getPosOjoDer(landmarks)

        eyeWidth = int(hypot(leftEye1[0] - rightEye1[0], leftEye1[1] - ri
ghtEye1[1]) * 2)
        eyeHeight = int(eyeWidth * 1)

        # Se toma la esquina superior izq para el cálculo del área
        top_left1 = (int(centerEye1[0] - eyeWidth / 2),
                     int(centerEye1[1] - eyeHeight / 2))

        top_left2 = (int(centerEye2[0] - eyeWidth / 2),
                     int(centerEye2[1] - eyeHeight / 2))

        # Se cambia el tamaño de la imagen para que se adapte
        dollarEye = cv.resize(eye_image, (eyeWidth, eyeHeight))
        dollarEyeGray = cv.cvtColor(dollarEye, cv.COLOR_BGR2GRAY)
        _, eyeMask = cv.threshold(dollarEyeGray, 25, 255, cv.THRESH_BINAR
Y_INV)

        # Se obtiene el área que va a ocupar la imagen en los ojos
        eyeArea1 = getEyeArea(top_left1, eyeHeight, eyeWidth, eyeMask)
        eyeArea2 = getEyeArea(top_left2, eyeHeight, eyeWidth, eyeMask)

        # Se coloca la imagen en ambos ojos
        frame[top_left1[1]: top_left1[1] + eyeHeight,
              top_left1[0]: top_left1[0] + eyeWidth] = cv.add(eyeArea1, d
ollarEye)

        frame[top_left2[1]: top_left2[1] + eyeHeight,
```

```
            top_left2[0]: top_left2[0] + eyeWidth] = cv.add(eyeArea2, d
ollarEye)

    cv.imshow("DLIB", frame)

    # Pulsar 'Esc' para salir del bucle
    key = cv.waitKey(1)
    if key == 27: break
```

10.3. Ejemplos

El resultado del filtro se puede comprobar en las siguientes capturas tomadas desde la cámara. Como puede verse, el símbolo del dólar se coloca correctamente en los ojos y se mantiene, aunque se mueva la cabeza:





11. Ejercicio “NFACE”

11.1. Enunciado

Implementa la normalización de caras explicada en la sección de transformaciones afines.

11.2. Resolución

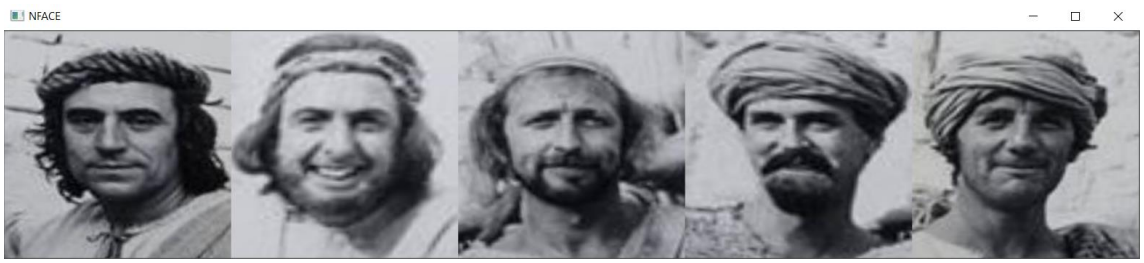
Para este ejercicio, la idea es tomar 3 puntos para realizar una transformación del plano afín de las caras de la imagen de ejemplo. Estos 3 puntos provienen de la imagen mostrada en el ejercicio anterior y son los extremos exteriores de los ojos y la parte superior central de la boca. Es un ejercicio parecido a la rectificación de planos de “RECTIF”, pero esta es afín como he comentado. Una vez establecidos los puntos, se obtiene la matriz de transformación y se normalizan las caras con “warpAffine()”. Las caras se detectan con “detector” y cuando se normalizan se colocan horizontalmente para mostrarlas en pantalla.

```
import sys
import dlib
import imutils
import cv2          as cv
import numpy        as np

from imutils        import face_utils
from umucv.stream   import autoStream


# Función que devuelve una cara normalizada según la posición exterior de
 los ojos
# y la parte superior de la boca

def normalizarCara(cara, landmarks):

    # Las referencias son los extremos de los ojos y la parte superior de
 la boca
    referencias = np.float32(landmarks[np.array([36, 45, 62])])
    # Se establecen los ojos a la misma altura y la boca entre ambos
    nuevaPosicion = np.float32([[80,100],[160,100],[120,150]])

    # Se obtiene la matriz de transformación
    matriz = cv.getAffineTransform(referencias,nuevaPosicion)

    # Se normaliza la cara con la matriz de transformación
    caraNormalizada = cv.warpAffine(cara, matriz, (250, 250))
    return caraNormalizada
```

```
# Se carga el predictor
pathPredictor = "./dlib/shape_predictor_68_face_landmarks.dat"
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(pathPredictor)

# Se lee la imagen
imagen = cv.imread(sys.argv[1]) # "./nface/monty-python1.jpg"
gray = cv.cvtColor(imagen, cv.COLOR_BGR2GRAY)

# Se detectan las caras de la imagen
dets = detector(gray, 1)
caras = []

# Se itera sobre las detecciones de la cara
for det in dets:

    # Se toman los 'landmarks' de la cara actual
    landmarks = predictor(gray, det)
    landmarks = face_utils.shape_to_np(landmarks)

    # Se añade la normalización de esa cara usando los
    # 'landmarks' a la lista de caras
    caras.append(normalizarCara(imagen, landmarks))

# Se combinan las caras en fila en una sola imagen
carasApiladas = np.hstack(caras)

# Se puede salir del programa pulsando la tecla 'Esc'
while(True):
    key = cv.waitKey(1) & 0xFF
    if key == 27: break
    cv.imshow("NFACE", carasApiladas)
```

11.3. Ejemplos

El resultado de este ejercicio sobre la imagen de los Monty Python es el siguiente:



Puede verse que los ojos de las caras están a la misma altura, y la boca centrada con ambos.